° curso / 2° cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas. Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Alberto Jesús Durán López

Grupo de prácticas: 1

Fecha de entrega:

Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz

Sistema operativo utilizado: Ubuntu 14.04.1 - 64 bits

Versión de gcc utilizada: gcc version 4.9.4 (gcc -v)

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

- 1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables globales):
 - 1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use -O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.
 - 1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.
 - 1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

A) MULTIPLICACIÓN DE MATRICES:

CÓDIGO FUENTE: pmm-secuencial.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000

int **matrizA, **matrizB, **matrizC;

int main(int argc, char** argv)
{
   int i, j, k;
   int valor;
   int suma=0;
```

```
if (argc<2){
   printf("Introduzca tamaño de la matriz\n");
   exit(-1);
  }
 int N = atoi(argv[1]);
  if(N>MAX)
     N=MAX;
 matrizA=(int **)malloc(N*sizeof(int*));
 matrizB=(int **)malloc(N*sizeof(int*));
 matrizC=(int **)malloc(N*sizeof(int*));
 for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
 srand (time(NULL));
  for (i=0; i<N; i++){
   for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
      matrizA[i][j]=valor;
      matrizB[j][i]=valor;
   }
 }
  struct timespec cgt1,cgt2; double ncgt;
 clock_gettime(CLOCK_REALTIME, &cgt1);
  //Calculamos el producto de las matrices
 for (i=0; i<N; i++){
   for(j=0; j<N; j++){
        for (k=0; k<N; k++){}
          matrizC[i][j]+= matrizA[i][k]*matrizB[k][j];
                    }
 clock_gettime(CLOCK_REALTIME, &cgt2);
 //Calculamos la diferencia
 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
  printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);
  //Liberamos memoria
 for(i=0; i<N; i++){
   free(matrizA[i]);
   free(matrizB[i]);
    free(matrizC[i]);
  }
 free(matrizA);
```

```
free(matrizB);
free(matrizC);

return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –**explicación-:** Realizo un cambio en la ejecución de los bucles for anidados. En vez de hacer (i,j,k) hago (i,k,j) para acceder a elementos consecutivos al llamar a MatrizB[k][j]

Modificación b) –explicación-: He realizado un desenrollado del bucle, realizando saltos de 10 iteraciones para reducirlas.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) pmm-secuencial-modificado_a.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000
int **matrizA, **matrizB, **matrizC;
int main(int argc, char** argv)
 int i, j, k;
 int valor;
 int suma=0;
  if (argc<2){
    printf("Introduzca tamaño de la matriz\n");
    exit(-1);
 int N = atoi(argv[1]);
  if(N>MAX)
     N=MAX;
 matrizA=(int **)malloc(N*sizeof(int*));
 matrizB=(int **)malloc(N*sizeof(int*));
 matrizC=(int **)malloc(N*sizeof(int*));
 for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
 }
  srand (time(NULL));
 for (i=0; i<N; i++){
    for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
     matrizA[i][j]=valor;
      matrizB[j][i]=valor;
   }
  }
```

```
struct timespec cgt1,cgt2; double ncgt;
 clock_gettime(CLOCK_REALTIME, &cgt1);
  //Calculamos el producto de las matrices
 for (i=0; i<N; i++){
    for(k=0; k<N; k++){
        for (j=0; j<N; j++){
          matrizC[i][j]+= matrizA[i][k]*matrizB[k][j];
                   }
  }
 clock_gettime(CLOCK_REALTIME, &cgt2);
 //Calculamos la diferencia
 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
 printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);
 //Liberamos memoria
 for(i=0; i<N; i++){
   free(matrizA[i]);
   free(matrizB[i]);
   free(matrizC[i]);
 }
 free(matrizA);
 free(matrizB);
 free(matrizC);
 return 0;
}
```

b) pmm-secuencial-modificado_b.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 3000

int **matrizA, **matrizB, **matrizC;

int main(int argc, char** argv)
{
    int i, j, k;
    int valor;
    int suma=0;

    if (argc<2){
        printf("Introduzca tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);</pre>
```

```
if(N%10!=0) {
   fprintf(stderr, "Num debe ser divisible entre 10\n");
   exit(-1);
 }
  if(N>MAX)
     N=MAX;
 matrizA=(int **)malloc(N*sizeof(int*));
 matrizB=(int **)malloc(N*sizeof(int*));
 matrizC=(int **)malloc(N*sizeof(int*));
  for(i=0; i<N; i++){
    matrizA[i]=(int*)malloc(N*sizeof(int));
    matrizB[i]=(int*)malloc(N*sizeof(int));
    matrizC[i]=(int*)malloc(N*sizeof(int));
  srand (time(NULL));
  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
      valor=rand() % 1000; //Número aleatorio entre 0 y 999
      matrizA[i][j]=valor;
      matrizB[j][i]=valor;
    }
  }
  int t0, t1, t2, t3, t4, t5, t6, t7, t8, t9;
  struct timespec cgt1,cgt2; double ncgt;
  clock_gettime(CLOCK_REALTIME, &cgt1);
  //Calculamos el producto de las matrices
  for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        t0=t1=t2=t3=t4=t5=t6=t7=t8=t9=0;
        for (k=0; k<N; k+=10){
          t0+=matrizA[i][k]*matrizB[k][j];
          t1+=matrizA[i][k+1]*matrizB[k+1][j];
          t2+=matrizA[i][k+2]*matrizB[k+2][j];
          t3+=matrizA[i][k+3]*matrizB[k+3][j];
          t4+=matrizA[i][k+4]*matrizB[k+4][j];
          t5+=matrizA[i][k+5]*matrizB[k+5][j];
          t6+=matrizA[i][k+6]*matrizB[k+6][j];
          t7+=matrizA[i][k+7]*matrizB[k+7][j];
          t8+=matrizA[i][k+8]*matrizB[k+8][j];
          t9+=matrizA[i][k+9]*matrizB[k+9][j];
         matrizC[i][j]=t0+t1+t2+t3+t4+t5+t6+t7+t8+t9;
 clock_gettime(CLOCK_REALTIME, &cgt2);
  //Calculamos la diferencia
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
```

```
printf("Tiempo(seg.):%f\t / Tamaño:%u\t/ \n", ncgt, N);

//Liberamos memoria
for(i=0; i<N; i++){
   free(matrizA[i]);
   free(matrizB[i]);
   free(matrizC[i]);
}

free(matrizA);
free(matrizB);
free(matrizC);

return 0;
}</pre>
```

Capturas de pantalla (que muestren que el resultado es correcto):

Aunque no lo pedían, he realizado un script para mostrar los resultados para un N=500 y optimización -02. Así que en una única captura mostraré todos los resultados de los tiempos.

```
albduranlopez@albduranlopez: ~/Escritorio/ultima entrega AC/codigo
albduranlopez@albduranlopez: ~/Escritorio/ultima entrega AC/codigo$ ./script_matrices.sh
Código normal. Optimización -O2
Tiempo(seg.):0.133183 / Tamaño:500 /
Código optimizado A. Optimización -O2
Tiempo(seg.):0.086194 / Tamaño:500 /
Código optimizado B. Optimización -O2
Tiempo(seg.):0.124823 / Tamaño:500 /
albduranlopez@albduranlopez:~/Escritorio/ultima entrega AC/codigo$
```

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0.133183
Modificación a)	0.086194
Modificación b)	0.124823

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Podemos observar claramente que el código sin modificar obtiene peores tiempos que las modificaciones realizadas con la optimización pedida -02. En este caso, la modificación a) es la que tiene el mejor tiempo de todos, 0.086194 segundos en realizar el producto de dos matrices 500*500. Es decir, realizar un cambio en la ejecución de los bucles for anidados. En vez de hacer (i,j,k), hacer(i,k,j).

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP): (BONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA LISE

(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secu	encial.s	pmm-secu modifica		pmm-secu	
call	clock_gettime	call	clock_gettime	call	clock_gettime
	movq		movq		movq
	matrizC(%rip), %r13	av de	matrizC(%rip),		matrizA(%rip), %ra
	movq	%rbp			movq
	matrizA(%rip), %r12		movq		\$0, 32(%rsp)
	movq	0/44	matrizA(%rip),		movq
.L11:	matrizB(%rip), %r11	%r11	xorl		%rax, 40(%rsp)
. L11.	mova		%r10d, %r10d		movq matrizB(%rip), %ra
	movq 0(%r13,%rbp,8),		movq		movq
%rdx	σ(%i ±3, %i bp, σ),		matrizB(%rip),		%rax, 24(%rsp)
701 GX	movq	%r9	maci 125(%i 1p),	.L12:	701 dx, 24(701 3p)
	(%r12,%rbp,8), %r10	.L11:			movq
	xorl		movq		40(%rsp), %rax
	%r8d, %r8d		0(%rbp,%r10,8),		movq
	.p2align 4,,10	%rsi	· (···· -p/····///		32(%rsp), %rbx
	.p2align 3		movq		movq
.L14:	. F9 0		(%r11,%r10,8),		\$0, 8(%rsp)
	movl	%r8	(- / / - / /		movq
	(%rdx,%r8,4), %esi		xorl		(%rax,%rbx,8), %ra
	leag		%edi, %edi		movq
	0(,%r8,4), %r9		.p2align 4,,10		%rax, 16(%rsp)
	xorl		.p2align 3		.p2align 4,,10
	%eax, %eax	.L14:	-		.p2align 3
	.p2align 4,,10		movq	.L15:	_
	.p2align 3		(%r9,%rdi,8),		movq
.L9:		%rdx			8(%rsp), %rax
	movq		xorl		movq
	(%r11,%rax,8), %rdi		%eax, %eax		16(%rsp), %rdx
	movl		.p2align 4,,10		xorl
	(%r10,%rax,4), %ecx		.p2align 3		%edi, %edi
	addq	.L9:			xorl
	\$1 , %rax		movl		%r15d, %r15d
	imull		(%r8,%rdi,4),		xorl
	(%rdi,%r9), %ecx	%ecx			%r14d, %r14d
	addl		imull		xorl
	%ecx, %esi		(%rdx,%rax,4),		%r13d, %r13d
	cmpl	%ecx	. 4.42		xorl
	%eax, %ebx		addl		%r12d, %r12d
	mov1	0(4)	%ecx, (%rsi,		xorl
	%esi, (%rdx,%r8,4)	%rax,4)	adda		%ebp, %ebp
	jg .L9		addq \$1, %rax		xorl %ahy %ahy
	. L9 addq		\$1, %rax cmpl		%ebx, %ebx leag
	\$1, %r8		%eax, %ebx		0(,%rax,4), %rcx
	cmpl				movq
	%r8d, %ebx		jg .L9		24(%rsp), %rax
	jg		addq		xorl
	.L14		\$1, %rdi		%r11d, %r11d
	addq		cmpl		xorl
	\$1, %rbp		%edi, %ebx		%r10d, %r10d
	cmp1		jg		xorl
	%ebp, %ebx		. L14		%r9d, %r9d
	jg		addq		xorl
	.L11		\$1, %r10		%r8d, %r8d
	leaq		cmpl		movl
	16(%rsp), %rsi		%r10d, %ebx		%edi, (%rsp)
	xorl		jg		.p2align 4,,10
	%edi, %edi		.L11		.p2align 3
	xorl		leag	.L10:	. 5
	%ebp, %ebp		16(%rsp), %rsi		movq
	call		xorl		(%rax), %rsi
	clock_gettime	1	%edi, %edi	I	movl

16(%rsp), %rax %cbp, %ebp call sid, %rsp sid			-		(0/-1) 0/-12
Subq		•	xorl		(%rdx), %edi
(Mrsp), Weak Mobb, Weak M					
Moving		= -			
Sebs, Werk Sem		_	clock_gettime		
Decoration Semilar S					
Semin Semin Semin		'			
S.1C2, NoSi					
S.LCZ, Wesi pxpr 72(%rax), %rsi and movq movi with west, %rsi,		,			
PXOT SkmmB, SkmmB movi mill					
Normal N		·		72(%rox) %roi	illovq -
mov1 %1, %edi cvtsi2sedq mov1 %2 mov1 %2 mov1 mov2 mov2 mov3 mov4 m				12(%1ax), %151	addl
S1, Medi					
CVS12Sdq Sfax, Skem1 movq 24(%rsp), %rax subt switch					
SFRX, SKEMEL INUIT				36(%rdv) %edi	1110 V I
movq 24(%rsp), %rax subq 8(%rsp), %rax subq 8(%rsp), %rax cvtsi2sdq movq movq movq movq movq movq movq size, %red movq size, %red movq size, %red movq movq matriz(%rip), %rax movq (%rax, %rbp, 8), %rdi call free movq matriz(%rip), %rax movq (%rax, %rbp, 8), %rdi dadd si, %rid movq movq (%rax, %rbp, 8), %rdi dadd si, %rid dadd, %rid movq si, %rid dadd si, %rid movq si, %rid				30(%idx), %edi	imull
24(%rsp), %rax subq 8(%rsp), %rax cvtsi25dq		·			
Subp 64(%rax), %rsi add1 %edi, %r9d movd					
S(%rsp), %rax cvtsized %rax, %xmm0 %edi, %rad %edi, %rad %edi, %rad %edi, %rad %edi, %rad %edi, %rad mov1 mult (wrsi, %racx), %edi mov1 mult (wrsi, %racx), %edi mov2 mov4 mo				64(%rax), %rsi	mo v q
Cvtsi2sdq				0.(10.41), 10.01	addl
Marax, %mme mov1 32(%rdx), %edi imul (%rsi,%rcx), %edi divsd (\dotsc)					
mov1		·			
S1, %eax dived (%rsi, %rcx), %edi (%rsi, %rcx		'		32(%rdx), %edi	
divsd LC1(%rip), %xmme addsd S6(%rax), %rsi add S6(%rax), %rsi add Sedi, %riod movulation S6(%rax), %rsi add S6(%rax), %rsi				,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	imull
addsd Sxmmn, %xmm0 call		-			(%rsi,%rcx), %edi
addsd Sxmmn, %xmm0 call					, , , , , , , , , , , , , , , , , , , ,
Sxmm1, Sxmm0 call addl addl movl call movq mart1zA(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq (%rax,%rbp,8), %rdi addl movl movq (%rax,%rbp,8), %rdi addl movl movq (%rax,%rbp,8), %rdi addl movq (%rax,%rbp,8), %rdi addl movq (%rax,%rbp,8), %rdi addl movq (%rax,%rbp,8), %rdi addl (%rsi,%rcx), %edi movq (%rax,%rbp,8), %rdi addl %edi,%rsi,%rcx), %edi movq (%rax,%rbp,8), %rdi call free movq martizA(%rip), %rdi call free movq martizA(%rip), %rdi call free movq martizA(%rip), %rdi call free addq %40, %rsp .cfi,remember_state .cfl.def_cfa_offset cfl.def_cfa_offset fimull (%rsi,%rcx), %edi movq fimull (%rsi,%rcx), %edi movq fimull (%rsi,%rcx), %edi movq fimull (%rsi,%rcx), %edi movq fimull (%rsi,%rcx), %edi fimull (%rsi,%rcx), %edi movq fimull (%rsi,%rcx), %edi fimull fimull (%rsi,%rcx), %edi fimull f				56(%rax), %rsi	•
		%xmm1, %xmm0		, ,,	addl
L13: movq matrizA(%rip), %rax movq matrizA(%rip), %rax movq matrizA(%rip), %rax movq					
1.13: movq matrizA(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizB(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizA(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi addl %edi, %ebx movl free cmp1 %ebp, %ebx jo 16(%rdx), %edi movq movq matrizA(%rip), %rdi call free movq matrizA(%rip), %rdi call free movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq movq fowq free movq movq fowq free movq fowq fo		printf_chk			
matrizA(%rip), %rax movq	.L13:			28(%rdx), %edi	
movq		movq			imull
(%rax,%rbp,8), %rdi call free movq matrizB(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi addq \$1, %rbp call free cmpl %ebp, %ebx jg Jg J.L13 .L15: movq matrizC(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free movq matrizC(%rip), %rdi call free movq matrizC(%rip), %rdi call free movq movd matrizC(%rip), %rdi call free movq mov1 18(%rax), %rsi mov1 18(%rdx), %edi mov1 18(%rax), %rsi addl %edi, %r13d mov1 S(%rax), %rsi mov1 S(%rax), %rsi mov2 S(%rax), %rsi addl %edi, %r14d mov4 S(%rax), %rsi addl %edi, %r12d mov1 S(%rax), %rsi addl %edi, %r12d mov1 S(%rax), %rsi addl %edi, %r12d mov1 S(%rax), %rsi		matrizA(%rip), %rax			(%rsi,%rcx), %edi
Call free movq matrizB(%rip), %rax movq (%rax, %rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax, %rbp,8), %rdi movq matrizC(%rip), %rax movq (%rax, %rbp,8), %rdi addq %edi, %ebx movq (%rax, %rbp,8), %rdi addq %edi, %ebx movq (%rax, %rbp,8), %rdi addq %edi, %ebp movq (%rax, %rbp,8), %rdi addq %edi, %ebp movq matrizC(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi movq free free movq matrizC(%rip), %rdi movq free		movq			movq -
Free		(%rax,%rbp,8), %rdi		48(%rax), %rsi	
movq matrizB(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi addd si, %rbp call free cmpl %ebp, %ebx jg lg late late late late late late late late		call			addl
matrize(%rip), %rax movq (%rax,%rbp,8), %rdi call free movq (%rax,%rbp,8), %rdi call free movq (%rax,%rbp,8), %rdi add movq (%rax,%rbp,8), %rdi addq %rdi, %edi movq (%rax,%rbp,8), %rdi addq \$1, %rbp call free cmpl %ebp, %ebx jg .li3		free			
movq		•			movl -
(%rax,%rbp,8), %rdi call (%rax,%rbp,8), %rdi movq free movq matrizC(%rip), %rax movq (%rax,%rbp,8), %rdi addq 40(%rax), %rdi addl %edi, %ebx movl \$1, %rbp call imull (%rsi,%rcx), %edi movq \$2(%rdx), %rdi addq 32(%rax), %rsi movq \$1, %rbp call addl %edi, %ebx movl \$2(%rax), %rsi movq addl %edi, %ebp movl \$1g .L13 .L13 .L15: 16(%rdx), %edi movl \$2(%rax), %rsi movq addl (%rsi,%rcx), %edi movl \$24(%rax), %rsi movq addl %edi, %r12d movl \$4(%rax), %rsi movq 12(%rdx), %edi movl \$4(%rax), %rsi movq 16(%rax), %rsi movq \$40, %rsp .cfi_remember_state .cfi_def_cfa_offset 8(%rdx), %edi movq \$6 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 8(%rax), %rsi addl %edi, %r14d movq				24(%rdx), %edi	
Call free movq 40(%rax), %rsi addl %edi, %ebx movd (%rax, %rbp, 8), %rdi addq (%rsi, %rcx), %edi movq (%rsi, %rcx), %edi (%rsi, %r					
free movq matrizC(%rip), %rax movq matrizC(%rip), %rax movq 20(%rdx), %edi imull (%rsi,%rcx), %edi movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free					
movq matrizC(%rip), %rax movq (%rax, %rbp, 8), %rdi addq si, %rbp call free cmpl webp, %ebx jg				10(0(movq -
matrizC(%rip), %rax movq				40(%rax), %rsi	. 443
movq		•			
(%rax,%rbp,8), %rdi addq \$1, %rbp call free cmpl %ebp, %ebx jg .l13 .l13 .l15: movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free movq mov1 12(%rdx), %edi imull (%rsi,%rcx), %edi movq 8(%rdx), %rsi imull (%rsi,%rcx), %edi movq 8(%rdx), %rsi addl %edi, %r13d movq 8(%rdx), %rsi addl %edi, %r14d movq 8(%rax), %rsi addl %edi, %r14d movq					
addq \$1, %rbp (%rsi, %rcx), %edi movq 32(%rax), %rsi addl				20(%rdy) %odi	
\$1, %rbp call free cmpl %ebp, %ebx jg .L13 .L15: movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free dodd matrizC(%rip), %rdi call free movq matrizC(%rip), %rdi call free addd \$40, %rsp .cfi_remember_state .cfi_def_cfa_offset 56 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 8(%rax), %rsi imull (%rsi,%rcx), %edi movq 8(%rax), %edi imull (%rsi,%rcx), %edi movq 8(%rax), %rsi addl %edi, %r13d movl 8(%rdx), %edi imull (%rsi,%rcx), %edi movq addl %edi, %r13d movl 8(%rdx), %edi movq addl %edi, %r13d movq 8(%rax), %rsi addl %edi, %r14d movq addl %edi, %r14d movq				20(%iux), %eui	imull
Call free cmpl webp, %ebx jg addl %edi, %ebp movl l6(%rdx), %edi webp, %ebx jg l6(%rdx), %edi l6(%rdx), %edi l6(%rdx), %edi l6(%rdx), %edi movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free ddq s40, %rsp cfi_remember_state cfi_def_cfa_offset l6(%rax), %rsi l6(%rax), %rsi movq l6(%rax), %rsi l6(%rax), %rsi l6(%rax), %edi movq l6(%rax), %rsi l6(%rax), %edi l6(%rax), %rsi l6(%rax), %edi l6(%ra					
free cmp1					
cmpl				32(%rax). %rsi	
Mebp, %ebx jg				- (3// / / / / / / / / / / / / / / / /	addl
Jg					
.L13 .L15: movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free movq matrizC(%rip), %rdi call free addq s40, %rsp .cfi_remember_state .cfi_def_cfa_offset xorl %eax, %eax popq %rbx .cfi_def_cfa_offset					
L15: movq matrizA(%rip), %rdi call (%rsi,%rcx), %edi movq (%rsi,%rcx), %edi movq (%rsi,%rcx), %edi movq (%rsi,%r12d movl (%rsi,%rcx), %edi movl (%rsi,%rcx), %edi movq (%rsi,%rcx), %edi (%rsi,%rcx), %edi (%rsi,%rcx), %edi (%rsi,%rcx), %edi (%rsi,%rcx), %edi movq (%rsi,%rcx), %edi (%r				16(%rdx), %edi	
movq matrizA(%rip), %rdi call free movq matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free addq s40, %rsp .cfi_remember_state .cfi_def_cfa_offset 12(%rdx), %rsi movl 12(%rdx), %edi movl 16(%rax), %rsi movq 16(%rax), %rsi movl 8(%rdx), %edi (%rsi,%rcx), %edi movl 8(%rdx), %edi (%rsi,%rcx), %edi movl 8(%rdx), %edi (%rsi,%rcx), %edi movl 8(%rdx), %rsi movq 8(%rax), %rsi movq	.L15:			` ',	imull
matrizA(%rip), %rdi call free movq movq addl %edi, %r12d movl movl imull (%rsi,%rcx), %edi movl imull (%rsi,%rcx), %edi movq imull (%rsi,%rcx), %edi movl imull (%rsi,%rcx), %edi movl imull (%rsi,%rcx), %edi movq imull (%rsi,%rcx), %edi m		movq			(%rsi,%rcx), %edi
free movq matrizB(%rip), %rdi call free imull (%rsi,%rcx), %edi movq matrizC(%rip), %rdi call free imull (%rsi,%rcx), %edi movq icall free addq & addq icali free icfi_remember_state icfi_def_cfa_offset imull (%rsi,%rcx), %edi icfi_def_cfa_offset imull icfi_def_cf		matrizA(%rip), %rdi			
movq %edi, %r12d matrizB(%rip), %rdi 12(%rdx), %edi free imull movq movq matrizC(%rip), %rdi movq call free addq %edi, %r13d \$40, %rsp movl .cfi_remember_state 8(%rdx), %edi .cfi_def_cfa_offset imull xorl movq %eax, %eax movq popq addl %rbx .cfi_def_cfa_offset				24(%rax), %rsi	
matrizB(%rip), %rdi call free movq matrizC(%rip), %rdi call free daddq suddq suddqsq suddqsq suddqsq suddqsq suddqsq suddqsq suddqsq suddqsq suddqsq s		free			addl
call free movq matrizC(%rip), %rdi call free addq \$40, %rsp .cfi_remember_state .cfi_def_cfa_offset xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 12(%rdx), %edi imull (%rsi,%rcx), %edi movq 16(%rax), %rsi 8(%rdx), %edi imull (%rsi,%rcx), %edi imull (%rsi,%rcx), %edi movq 8(%rax), %rsi addl %edi, %r14d %edi, %r14d movl		•			
free movq imatrizC(%rip), %rdi call free addq s40, %rsp .cfi_reemember_state .cfi_def_cfa_offset		matrizB(%rip), %rdi			movl -
movq (%rsi,%rcx), %edi matrizC(%rip), %rdi novq call 16(%rax), %rsi free addl s40, %rsp movl .cfi_remember_state 8(%rdx), %edi .cfi_def_cfa_offset imull xorl movq %eax, %eax movq popq addl %rbx addl .cfi_def_cfa_offset %edi, %r14d				12(%rdx), %edi	
matrizC(%rip), %rdi call free addq \$40, %rsp .cfi_remember_state .cfi_def_cfa_offset xorl %eax, %eax popq %rbx .cfi_def_cfa_offset movq 16(%rax), %rsi 8(%rdx), %edi movl 8(%rdx), %edi (%rsi,%rcx), %edi movq 8(%rax), %rsi addl %edi, %r14d %edi, %r14d movl					
call 16(%rax), %rsi free addl addl %edi, %r13d movl 8(%rdx), %edi imull (%rsi, %rcx), %edi xorl movq %eax, %eax movq popq addl %rbx cfi_def_cfa_offset		•			(%rsi,%rcx), %edi
free addq					movq -
addq				16(%rax), %rsi	. 4.45
\$40, %rsp .cfi_remember_state .cfi_def_cfa_offset xorl %eax, %eax popq %rbx .cfi_def_cfa_offset xorl %eax, %eax popq %rbx .cfi_def_cfa_offset \$8(%rdx), %edi (%rsi,%rcx), %edi movq 8(%rax), %rsi addl %edi, %r14d movl					
.cfi_remember_state 8(%rdx), %edi .cfi_def_cfa_offset imull 56 (%rsi,%rcx), %edi xorl movq %eax, %eax 8(%rax), %rsi popq addl %rbx .cfi_def_cfa_offset .cfi_def_cfa_offset movl		•			
.cfi_def_cfa_offset imull 56 (%rsi,%rcx), %edi xorl movq %eax, %eax 8(%rax), %rsi popq addl %rbx %edi, %r14d .cfi_def_cfa_offset movl				0 (0(64)) 0(-4)	movi -
56 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset (%rsi,%rcx), %edi movq 8(%rax), %rsi addl %edi, %r14d movl				o(%rux), %edl	imull
xorl	56	.cri_uer_cra_orrset			
%eax, %eax popq srbx .cfi_def_cfa_offset %eax, %eax 8(%rax), %rsi addl %edi, %r14d movl	30	vorl			
popq addl %rbx %edi, %r14d .cfi_def_cfa_offset movl				8(%ray) %roi	ovq -
%rbx %edi, %r14d .cfi_def_cfa_offset movl				ο(%ιαΧ), %ΓS1	Ibbe
.cfi_def_cfa_offset movl					
41/01 UA 1, /0CUL	48	. GI I_UEI_GI a_UII SEL		4(%rdy) %adi	VI -
	70			-(/oi u/), /oeul	

	popq		imull
	%rbp		(%rsi,%rcx), %edi addl
40	.cfi_def_cfa_offset		%edi, %r15d
	popq		cmpl
	%r12		4(%rsp), %r8d
00	.cfi_def_cfa_offset		jl
32	popq		.L10 movl
	%r13		(%rsp), %edi
	.cfi_def_cfa_offset		movq
24			8(%rsp), %rax
	popq %r14		addl %r9d, %edi
	.cfi_def_cfa_offset		addl
16			%edi, %r10d
	popq %r1F		addl
	%r15 .cfi_def_cfa_offset		%r10d, %r11d addl
8	10.1_0001.000		%r11d, %ebx
	ret		addl
.L3:	ofi roctors state		%ebx, %ebp
	.cfi_restore_state xorl		movq 16(%rsp), %rbx
	%edi, %edi		addl
	call		%ebp, %r12d
	time		addl
	movl %eax, %edi		%r12d, %r13d addl
	call		%r13d, %r14d
	srand		addl
	movq		%r14d, %r15d
	%rsp, %rsi xorl		movl %r15d, (%rbx,
	%edi, %edi	%rax,4)	701 13d, (701 bX,
	call	, ,	addq
	clock_gettime		\$1, %rax
			cmpl %eax, 4(%rsp)
			movq
			%rax, 8(%rsp)
			jg
			. L15 addq
			\$1, 32(%rsp)
			movq
			32(%rsp), %rax
			cmpl %eax, 4(%rsp)
			jg
			.L12
			leaq
			64(%rsp), %rsi xorl
			%edi, %edi
			xorl
			%ebx, %ebx call
			clock_gettime
			movq
			64(%rsp), %rax
			subq 48(%rsp), %rax
			48(%rsp), %rax movl
			\$.LC3, %esi
			pxor
			%xmm1, %xmm1 movl
			4(%rsp), %ebp
			pxor
			%xmm0, %xmm0
			movl \$1, %edi
			\$1, %ed1 cvtsi2sdq
			%rax, %xmm1
			movq
			72(%rsp), %rax
			subq 56(%rsp), %rax
			55(.0. 5p)/ max

		movl
		%ebp, %edx
		cvtsi2sdq
		%rax, %xmm0
		movl
		\$1, %eax
		divsd
		.LC2(%rip), %xmm0
		addsd
		%xmm1, %xmm0
		call
		printf_chk
	.L14:	
		movq
		matrizA(%rip), %rax
		movq
		(%rax,%rbx,8), %rdi
		call
		free
		movq
		matrizB(%rip), %rax
		movq
		(%rax,%rbx,8), %rdi
		call
		free
		movq
		matrizC(%rip), %rax
		movq
		(%rax,%rbx,8), %rdi
		addq
		\$1, %rbx
		call
		free
		cmpl
		%ebx, %ebp
		jg
		.L14
	.L16:	
		movq
		matrizA(%rip), %rdi
		call
		free
		movq
		matrizB(%rip), %rdi
		call
		free
		movq
		matrizC(%rip), %rdi
		call
		free
		addq
		\$88, %rsp
		.cfi_remember_state
		.cfi_def_cfa_offset
		. 51 1_G51_61 a_011 Set
	56	
		xorl
		%eax, %eax
		popq
		%rbx
		.cfi_def_cfa_offset
	48	. 5. 1_051_514_511561
	-0	nong
		popq
		%rbp
		.cfi_def_cfa_offset
	40	
		popq
		%r12
		.cfi_def_cfa_offset
	32	.571_661_614_611361
	32	202
		popq
		%r13
		.cfi_def_cfa_offset
	24	
		popa
		popq %r14
		%r14
	16	%r14 .cfi_def_cfa_offset
		<pre>%r14 .cfi_def_cfa_offset popq</pre>
		%r14 .cfi_def_cfa_offset

	_	.cfi_def_cfa_offset
	8	ret
		rec
	.L4:	
		.cfi_restore_state
		xorl
		%edi, %edi
		call
		time
		movl
		%eax, %edi
		call
		srand
		leaq
		48(%rsp), %rsi
		xorl
		%edi, %edi
		call
		clock_gettime

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: figura1-original.c

```
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
struct {
  int a;
  int b;
} s[5000];
int main(){
  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];
  srand(time(NULL));
  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  struct timespec cgt1,cgt2;
  double ncgt;
  clock_gettime(CLOCK_REALTIME, &cgt1);
  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i++)
      X1+=2*s[i].a+ii;
    for(i=0; i<5000;i++)
      X2+=3*s[i].b-ii;
    if (X1<X2)
      R[ii]=X1;
    else
      R[ii]=X2;
```

```
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Junto los dos bucles for en sólo uno ya que al fin y al cabo el bucle hace las mismas iteraciones y hace lo mismo.

Modificación b) –explicación-: Además de la modificación a) anterior, realizo un desenrollado del bucle para así decrementar el número de iteraciones realizadas y disminuir su tiempo de ejecución.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1-modificado_a.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
struct {
  int a;
  int b;
} s[5000];
int main(){
  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];
  srand(time(NULL));
  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  }
  struct timespec cgt1,cgt2;
  double ncgt;
  clock_gettime(CLOCK_REALTIME, &cgt1);
//Agrupo los dos for en uno solo
  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i++){
      X1+=2*s[i].a+ii;
      X2+=3*s[i].b-ii;
    if (X1<X2)
      R[ii]=X1;
    else
      R[ii]=X2;
  }
  clock_gettime(CLOCK_REALTIME, &cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-
```

```
cgt1.tv_nsec)/(1.e+9));
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}
```

b) figura1-modificado_b.c

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
struct {
  int a;
  int b;
} s[5000];
int main(){
  int MAX=40000;
  int ii, i;
  int X1, X2;
  int R[MAX];
  srand(time(NULL));
  for(i=0; i<5000; i++){
     s[i].a=rand()%5000;
     s[i].b=rand()%5000;
  struct timespec cgt1,cgt2;
  double ncgt;
  clock_gettime(CLOCK_REALTIME, &cgt1);
//Agrupo los dos for en uno solo y descompongo el bucle for
  for (ii=0; ii<40000;ii++){
    X1=0; X2=0;
    for(i=0; i<5000;i+=10){
      X1+=2*s[i].a+ii;
      X2+=3*s[i].b-ii;
      X1+=2*s[i+1].a+ii;
      X2+=3*s[i+1].b-ii;
      X1+=2*s[i+2].a+ii;
      X2+=3*s[i+2].b-ii;
      X1+=2*s[i+3].a+ii;
      X2+=3*s[i+3].b-ii;
      X1+=2*s[i+4].a+ii;
      X2+=3*s[i+4].b-ii;
      X1+=2*s[i+5].a+ii;
      X2+=3*s[i+5].b-ii;
      X1+=2*s[i+6].a+ii;
      X2+=3*s[i+6].b-ii;
      X1+=2*s[i+7].a+ii;
      X2+=3*s[i+7].b-ii;
      X1+=2*s[i+8].a+ii;
      X2+=3*s[i+8].b-ii;
      X1+=2*s[i+9].a+ii;
      X2+=3*s[i+9].b-ii;
```

```
if (X1<X2)
    R[ii]=X1;
    else
        R[ii]=X2;
}

clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("\nTiempo (seg.) = %11.9f\n", ncgt);
}</pre>
```

Capturas de pantalla (que muestren que el resultado es correcto):

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0.206034848
Modificación a)	0.142504281
Modificación b)	0.110772525

- **1.1. COMENTARIOS SOBRE LOS RESULTADOS:** Como podemos observar, el código sin modificar se ejecuta en un tiempo mayor que los otros dos modificados. Es por esto por lo que podemos observar que las modificaciones realizadas de unir los dos bucles for en uno son efectivas. Comparando la segunda modificación con la tercera, vemos que la modificación b) de desenrollado del bucle for tiene un tiempo menor por lo que también resulta efectiva dicha modificación
- 1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):

(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

Figura1-original.s		Figura1-modificado_a.s		Figura1-modificado_b.s	
call clock_c xorl %r8d, % .p2alic .p2alic	6r8d gn 4,,10	call	clock_gettime xorl %r8d, %r8d .p2align 4,,10 .p2align 3	call	clock_gettime xorl %r11d, %r11d .p2align 4,,10 .p2align 3

				T	
.L3:		.L2:		.L3:	
	movl		movl %r8d, %edi		movl %r11d, %r10d
	%r8d, %edi movl		movl		movl \$s,
	\$s, %eax		\$s, %eax	%eax	1110VI \$5,
	xorl		xorl	70CUX	xorl
	%esi, %esi		%ecx, %ecx		%edx, %edx
	.p2align 4,,10		xorl		xorl
	.p2align 3		%esi, %esi		%esi, %esi
.L4:			.p2align 4,,10		.p2align 4,,10
	movl		.p2align 3		.p2align 3
	(%rax), %edx	.L3:		.L4:	
	addq		movl		movl
	\$8, %rax		(%rax), %edx		(%rax), %ecx
	leal		addq		addq \$80,
	(%rdi,%rdx,2),		\$8, %rax	%rax	
%edx	- 441		leal		leal
	addl	0/ody	(%rdi,%rdx,2),		(%r10,%rcx,2), %r13d
	%edx, %esi	%edx	0.441	76 (%rov) %oov	movl -
	cmpq %ray %r12		addl %ody %osi	76(%rax), %ecx	addl
	%rax, %r12		%edx, %esi		addl %esi, %r13d
	jne		movl		•
	.L4 movl		-4(%rax), %edx leal		leal (%rcx,%rcx,2), %esi
	\$s+4, %eax		(%rdx,%rdx,2),		movl -
	xorl	%edx	(101 dA, 101 dA, 2),	72(%rax), %ecx	
	%ecx, %ecx	,scan	subl	/ L (// U.A.) , // // // // // // // // // // // // /	subl
	.p2align 4,,10		%edi, %edx		%r10d, %esi
	.p2align 3		addl		addl
.L5:	.pzarry o		%edx, %ecx		%esi, %edx
	movl		cmpq		leal
	(%rax), %edx		\$s+40000, %rax		(%r10,%rcx,2), %esi
	addq		jne		movl -
	\$8, %rax		.L3	68(%rax), %ecx	
	leal		cmpl	, ,,	addl
	(%rdx,%rdx,2),		%esi, %ecx		%esi, %r13d
%edx			cmovg		leal
	subl		%esi, %ecx		(%rcx,%rcx,2), %ecx
	%edi, %edx		movl		subl
	addl		%ecx, (%rbx,		%r10d, %ecx
	%edx, %ecx	%r8,4)			leal
	cmpq		addq		(%rcx,%rdx), %esi
	\$s+40004, %rax		\$1, %r8		movl -
	jne		cmpq	64(%rax), %edx	
	. L5		\$40000, %r8		leal
	cmpl		jne		(%r10,%rdx,2), %edx
	%esi, %ecx		.L2		addl
	cmovg		leaq		%edx, %r13d
	%esi, %ecx		-48(%rbp), %rsi		movl -
	movl		xorl	60(%rax), %edx	11
0/=0 4)	%ecx, (%rbx,		%edi, %edi		leal
%r8,4)	adda		call		(%rdx,%rdx,2), %edx
	addq \$1, %r8		clock_gettime		subl %r10d, %edx
	Cmpq				leal
	\$40000, %r8				(%rdx,%rsi), %ecx
	jne				mov1 -
	.L3			56(%rax), %edx	
	leag			00(701 007)	leal
	-64(%rbp), %rsi				(%r10,%rdx,2), %r9d
	xorl				mov1 -
	%edi, %edi			52(%rax), %edx	
	call			, ,,	addl
	clock_gettime				%r9d, %r13d
	- -				leal
					(%rdx,%rdx,2), %r9d
					subl
					%r10d, %r9d
					leal
					(%r9,%rcx), %edx
					movl -
				48(%rax), %ecx	
					leal
					(%r10,%rcx,2), %r9d
					movl -
				44(%rax), %ecx	- 443
					addl
I					%r9d, %r13d
					leal

Sublate Subl		
Srind, Sred Leal (%re, Srdx), %red Leal (%re, Srdx), %red Leal (%re, Srdx), %red Leal (%re, Srdx, 2), %red Leal (%rdx, S		(%rcx,%rcx,2), %r8d
Leal		
(%r-a, %rdx), %rdx 40(%rax), %edx 10al		
### 100 10		
leal (%10,%rdx,2), %r8d mov1 36(%rax), %edx 4dd 3763, %r3d (%rdx, %rdx,2), %eds mov1 32(%rax), %edx (%rdx, %rdx,2), %r9d mov1 28(%rax), %edx 4(%rax), %edx 24(%rax), %edx 24(%rax), %edx 3rd (%rdx, %rdx,2), %r9d mov1 24(%rax), %edx 3rd (%rdx, %rdx,2), %r9d mov1 26(%rax), %edx 3rd (%rdx, %rdx,2), %esi mov1 3rd (%rdx, %rdx,2), %edx mov1 3		
(%r10, %rdx, 2), %r8d mov1	40(%rax), %edx	loal
38(%rax), %edx add]		
36(%rax), %edx add1 %rdd, %r13d lea1 (%rdx, %rdx, 2), %edi mov1 32(%rax), %edx sub1 (%rdi, %r9), %r8d lea1 (%rdx, %rdx, 2), %r9d mov1 24(%rax), %edx sub1 %r13d, %r9d lea1 (%rdx, %rdx, 2), %r9d mov1 24(%rax), %edx sub1 %r13d, %r9d, %edi lea1 (%rdx, %rdx, 2), %r8d mov1 29(%rax), %edx sub1 %r13d, %r9d, %edi lea1 (%rdx, %rdx, 2), %r8d mov1 16(%rax), %edx sub1 %r13d, %rdx, 2), %r8d mov1 12(%rax), %edx sub1 %r13d, %rdx, 2), %edx mov1 12(%rax), %edx sub1 %r13d, %rdx, 2), %edx mov1 12(%rax), %edx lea1 (%rdx, %rdx, 2), %edx sub1 %r13d, %rdx lea1 (%rdx, %rdx, 2), %edx sub1 lea1 (%rdx, %rdx, 2), %edx sub1 lea1 (%rdx, %rdx, 2), %edx sub1 lea1 (%rdx, %r		
	36(%rax), %edx	
leal		
(%rcx, %rdx, 2), %edi movl		
32 (%rax), %edx sublimate		
Subl		
	32(%rax), %edx	
leal (%rd1,%r9), %red leal (%r10,%rdx,2), %r9d movl movl movl leal (%rdx,%rdx,2), %r9d movl leal (%rdx,%rdx,2), %esi movl leal (%rdx,%rdx,2), %esi movl leal (%r3,%r8), %edi leal (%r3,%r8), %edi leal (%r10,%rdx,2), %r8d movl leal (%rdx,%rdx,2), %r8d leal (%rdx,%rdx,2), %ecx movl leal (%rdx,%rdx,2), %ecx leal (%r0,%rdx), %edi leal (%r0,%rdx), %edi leal (%r0,%rdx), %edi leal (%r0,%rdx,2), %edi leal (%r0,%rdx,2), %edi leal (%rdx,%rdx,2), %edi leal (%rdx,%rdx,2), %edi leal (%rdx,%rdx,2), %edx leal (%rdx,xrdx,2), %edx leal (%rdx,x		
(%rd), %r9), %r8d leal (%r10, %rdx, 2), %r9d mov) 28(%rax), %edx addl %r13d, %r9d leal (%rdx, %rdx, 2), %esi mov) 24(%rax), %edx subl %r10d, %esi leal (%r10, %rdx, 2), %r8d mov) 20(%rax), %edx addl %r9d, %r8d, 2), %ecx mov) 16(%rax), %edx subl %r10d, %ecx leal (%rdx, %rdx, 2), %edi (%r10, %rdx, 2), %ecx mov) 12(%rax), %edx subl %r10d, %ecx leal (%rdx, %rdx, 2), %edx mov) 12(%rax), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx mov) 12(%rax), %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		
leal (%rie,%ridx,2), %red movl		
## addl ## add		leal
28(%rax), %edx addl %ri3d, %r9d leal (%rdx, %rdx, 2), %esi mov1 24(%rax), %edx subl %ri9d, %esi leal (%ri9, %rdx, 2), %edi leal (%ri9, %rdx, 2), %edi mov1 20(%rax), %edx subl %ri0d, %ecx leal (%rdx, %rdx, 2), %ecx mov1 16(%rax), %edx subl %ri0d, %ecx leal (%rdx, %rdx, 2), %edx ieal (%rdx, %rdx, 2), %edx subl %ri9d, %edi leal (%rdx, %rdx, 2), %edx subl %ri9d, %edx leal (%rdx, %rdx, 2), %edx subl %ri9d, %edx leal (%rdx, %rdx, 2), %edx subl %ri9d, %edx leal (%rdx, %rdx, 2), %edx leal (%rdx, xrdx, 2		
addl %ri3d, %r9d leal (%ridx, %rdx,2), %esi movl 24(%rax), %edx subl %ri3d, %esi leal (%ri3, %r8d), %esi leal (%ri4, %rdx,2), %r8d leal (%rdx, %rdx,2), %r8d leal (%rdx, %rdx,2), %ecx movl 16(%rax), %edx subl %ri3d, %r6d, %r6x leal (%rcx, %rdi), %esi leal (%rcx, %rdi), %esi leal (%rcx, %rdi), %esi leal (%rid, %rdx,2), %edx subl %ri3d, %edx leal (%rdx, %rdx,2), %edx le	20/0/==== 0/= 1	MOAT -
## ## ## ## ## ## ## ## ## ## ## ## ##	2δ(%r ax), %edx	addl
leal (%rdx,%rdx,2), %esi mov1 24(%rax), %edx 24(%rax), %edx		%r13d, %r9d
24(%rax), %edx Subland Subland Sibland		leal
24(%rax), %edx subl %riod, %esi loal (%rsi,%r8), %edi leal (%rsi,%r8), %edi leal (%rdx,%rdx,2), %r8d leal (%rox, %rdx,2), %ecx movl 16(%rax), %edx subl %riod, %ecx leal (%rcx, %rdi), %esi leal (%rcx, %rdx,2), %edi movl 12(%rax), %edx addl %riod, %ecx leal (%rcx, %rdi), %esi leal (%rox, %rdx,2), %edx subl %riod, %edx leal (%rox, %rdx,2), %edx subl %riod, %edx leal (%rox, %rdx,2), %edx subl %riod, %edx leal (%rdx, %rdx,2), %edx subl %riod, %edx addi %ecx, %edx cmpq %rax, %ri2		
Subl Wright, Wesi leal (%ris, %r8), %edi leal (%ris, %r8), %edi leal (%ris, %r8), %edi leal (%ris, %r6x, 2), %r8d movl subl %right, %r6x, 2), %ecx leal (%rdx, %rdx, 2), %edi leal (%ro, %rdi), %esi leal (%ris, %rdx, 2), %edi movl subl %right, %rdx, 2), %edx subl %right, %rdx, 2), %edx leal (%rdx, %rdx, 2), %edx leal	21/4504) 400	IIIOAT -
Wriod, Wesi leal (Wri, %r8), %edi leal (Wri, %r8), %edi leal (Wri, %r8), %edi leal (Wri, %r0x, 2), %r8d movl -	24(%rax), %edx	subl
leal (%rsi, %r8), %edi leal (%rsi, %r8), %edi leal (%r10, %rdx, 2), %r8d movl -		%r10d, %esi
leal (%r10,%rdx,2), %r8d mov1		leal
(%r10,%rdx,2), %r8d mov1 20(%rax), %edx add1 %r9d, %r8d leal (%rdx,%rdx,2), %ecx mov1 16(%rax), %edx sub1 %r10d, %ecx leal (%r10,%rdx,2), %edi mov1 12(%rax), %edx add1 %r8d, %edi leal (%r4x,%rdx,2), %edx sub1 %r10d, %edx leal (%rdx,%rsi), %ecx mov1 8(%rax), %edx 1eal (%r10,%rdx,2), %edx sub1 %r10d, %edx leal (%rfdx,%rsi), %ecx mov1 - 4(%rax), %edx 1eal (%r10,%rdx,2), %edx sub1 %r10d, %edx,2), %edx sub1 %r10d, %edx leal (%rdx,%rdx,2), %edx sub1 %r10d, %edx leal (%rdx,%rdx,2), %edx sub1 %r10d, %edx add1 %r10d, %edx add1 %r2d, %edx add1 %r2d, %edx add1 %r2d, %edx add1 %ecx, %edx cmpq %rax, %r12		
mov1 - add1		
29(%rax), %edx add1		
%r9d, %r8d leal (%rdx,%rdx,2), %ecx mov1	20(%rax), %edx	
leal (%rdx,%rdx,2), %ecx movl - 16(%rax), %edx subl		
(%rdx,%rdx,2), %ecx mov1 - 16(%rax), %edx subl %r19d, %ecx leal (%rcx,%rdi), %esi leal (%r10,%rdx,2), %edi mov1 - 12(%rax), %edx addl %r8d, %edi leal (%rdx,%rdx,2), %edx subl %r19d, %edx leal (%rdx,%rsi), %ecx mov1 - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx addl (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax,%r12		
16(%rax), %edx		
16(%rax), %edx subl %r10d, %ecx leal (%rcx,%rdi), %esi leal (%r10,%rdx,2), %edi mov1 12(%rax), %edx addl %r8d, %edi leal (%rdx,%rdx,2), %edx subl %r10d, %edx leal (%rdx,%rsi), %ecx mov1 - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %cx, %edx cmpq %rax, %r12		
### ##################################	16(%rax), %edx	
leal (%rcx,%rdi), %esi leal (%r10,%rdx,2), %edi movl - 12(%rax), %edx addl %r8d, %edi leal (%rdx,%rdx,2), %edx subl %r10d, %edx leal (%rdx,*rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%r10,%rdx,2), %edx leal (%r4x,xrdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %edx,%rdx,2), %edx addl %ecx, %edx cmpq %rax, %r12		
(%rcx,%rdi), %esi leal (%r10,%rdx,2), %edi movl - 12(%rax), %edx addl %r8d, %edi leal (%rdx,%rdx,2), %edx subl %r10d, %edx leal (%rdx,%rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		
leal (%r10,%rdx,2), %edi movl		
12(%rax), %edx add1		leal
12(%rax), %edx add1		
addl %*8d, %edi leal (%rdx, %rdx, 2), %edx subl %*10d, %edx leal (%rdx, %rsi), %ecx movl - 8(%rax), %edx leal (%r10, %rdx, 2), %edx leal (%rdx, %rdi), %esi movl - 4(%rax), %edx leal (%rdx, %rdx, 2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12	12/4ray) Vody	IIIOAT -
### ### ##############################	12(vi ax), weux	addl
(%rdx,%rdx,2), %edx subl %r10d, %edx leal (%rdx,%rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		%r8d, %edi
subl %r10d, %edx leal (%rdx,%rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		
### ##################################		
leal (%rdx,%rsi), %ecx mov1 - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx leal (%rdx,%rdx,2), %edx sub1 %r10d, %edx add1 %ecx, %edx cmpq %rax, %r12		
(%rdx,%rsi), %ecx movl - 8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		leal
8(%rax), %edx leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi mov1 - 4(%rax), %edx leal (%rdx,%rdx,2), %edx sub1 %r10d, %edx add1 %ecx, %edx cmpq %rax, %r12		(%rdx,%rsi), %ecx
leal (%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12	0/// \ 0/-	mov1 -
(%r10,%rdx,2), %edx leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12	8(%rax), %edx	leal
leal (%rdx,%rdi), %esi movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		
movl - 4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		leal
4(%rax), %edx leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		
leal (%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12	1/0/2011 0/04:	IIIOAT -
(%rdx,%rdx,2), %edx subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12	4(%1 ax), %edx	leal
subl %r10d, %edx addl %ecx, %edx cmpq %rax, %r12		(%rdx,%rdx,2), %edx
addl %ecx, %edx cmpq %rax, %r12		subl
%ecx, %edx cmpq %rax, %r12		
cmpq %rax, %r12		
%rax, %r12		
		%rax, %r12
jne .L4		jne .L4
cmpl		сшрт

```
%esi. %edx
                  cmovg
                  %esi,
                        %edx
                  movl
                  %edx, (%rbx,%r11,4)
                  addq
%r11
                  cmpq
                  $40000, %r11
                                     .L3
                  ine
                  lead
64(%rbp), %rsi
                  xorl
                  %edi,
                        %edi
                  call
                  clock_gettime
```

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada DAXPY (*Double precision- real Alpha X Plus Y*) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=1; i \le N, i++) y[i] = a*x[i] + y[i];
```

- 2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.
- 2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CÓDIGO FUENTE: daxpy.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

   double N=1000000000; //Tamaño del vector
   double a = 10, i; //Constante
   double *x, *y;

   x = (double*) malloc(N*sizeof(int));
   y = (double*) malloc(N*sizeof(int));

   for (i=0; i<N; i++){
        x[i] = i*5;
        y[i] = i/2+8;</pre>
```

```
struct timespec cgt1,cgt2; double ncgt;

clock_gettime(CLOCK_REALTIME,&cgt1);
//Becnhmark Linpack
for (i=0; i<N; i++)
    y[i] += a*x[i];

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("Tamaño: %i", N);
printf("\nTiempo (seg.) = %11.9f\n", ncgt);

free(x);
free(y);
return 0;
}</pre>
```

Tiempes eies	-O0	-O2	-O3
Tiempos ejec.	0.194425955	0.141920437	0.205654945

CAPTURAS DE PANTALLA:

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

- -O0: Desconecta por completo la optimización del código y es el predeterminado si no se especifica ningún -O. Es por esto por lo que el código en ensamblador de daxpyO0.c es más grande que el resto.
- -O2: suele ser la mejor opción de optimización, además, es el más recomendado. Se aumenta el rendimiento del código sin comprometer el tamaño y sin tomar mucho más tiempo de compilación.
- -O3: Aplica una optimización más agresiva pero puede defraudar. Activa optimizaciones que son caras en términos de tiempo de compilación y uso de memoria. No garantiza una mejora del rendimiento ya que puede ralentizar el sistema por lo que no se recomienda usarlo.

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP): (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpy	00. s	daxpy	02.s	daxpy	03.s
call	<pre>clock_gettime mov1 \$0, %ebx jmp</pre>	call %xmm1	<pre>clock_gettime movsd .LC1(%rip),</pre>	call %xmm1	<pre>clock_gettime movsd .LC1(%rip),</pre>
.L9:	.L8 movslq %ebx, %rax		xorl %eax, %eax .p2align 4,,10 .p2align 3		xorl %eax, %eax movsd .LC0(%rip),
	leaq 0(,%rax,8), %rdx movq -	.L4:	movsd (%rbx,%rax,8),	%xmm3	.p2align 4,,10 .p2align 3
64(%rbp), %rax	•	%xmm0	leal 1(%rax), %edx movsd .LCO(%rip), mulsd	.L4: %xmm0	movsd (%rbx,%rax,8), leal 1(%rax), %edx mulsd
64(%rbp), %rax	movq -	%xmm0	%xmm1, %xmm0 addsd (%r12,%rax,8), movsd	%×mm0	%xmm1, %xmm0 addsd (%r12,%rax,8), movsd
	(%rax), %xmm1 movslq %ebx, %rax leaq 0(,%rax,8), %rcx movq -	(%r12,%rax,8)	%xmm0, pxor %xmm0, %xmm0 addq \$1, %rax	(%r12,%rax,8)	<pre>%xmm0, pxor %xmm0, %xmm0 addq \$1, %rax</pre>
72(%rbp), %rax	addq %rcx, %rax movsd (%rax), %xmm0 mulsd -		cvtsi2sd %edx, %xmm0 ucomisd %xmm0, %xmm3 ja .L4		cvtsi2sd %edx, %xmm0 ucomisd %xmm0, %xmm3 ja .L4
80(%rbp), %xmm	n0 addsd %xmm0, %xmm1 movq %xmm1, %rax movq %rax, (%rdx) addl \$1, %ebx		<pre>leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime</pre>		<pre>leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime</pre>
.L8:	pxor %xmm0, %xmm0 cvtsi2sd %ebx, %xmm0				
88(%rbp), %xmm	movsd - n1 ucomisd %xmm0, %xmm1 ja .L9 leaq -				
32(%rbp), %rax					

|--|